SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR EXECUTING SCRIPTS ON MOBILE DEVICES

Inventors: David D. Kloba
Michael R. Gray
David M. Moore
Thomas E. Whittaker
David J. Williams
Rafael Z. Weinstein
Joshua E. Freeman
Linus M. Upson
Adam T. Dingle

This application is a continuation-in-part application of pending Serial No. 09/705,927, "System, Method, and Computer Program Product for a Scalable, Configurable, Client/Server, Cross-Platform Browser for Mobile Devices," filed on November 6, 2000 (Attorney Docket No. 1933.0010009), which is a continuation-in-part application of pending Ser. No. 09/559,964, "System, Method, and Computer Program Product for Enabling On-Device Servers, Offline Forms, and Dynamic Ad Tracking On Mobile Devices," filed April 28, 2000 (Atty. Docket No. 1933.0010001), which is a continuation-in-part application of pending Ser. No. 09/393,390, "Interactive Applications for Handheld Computers," filed September 10, 1999 (Atty. Docket No. 1933.0010000), and claims the benefit of U.S. Provisional Application No. 60/189,969, "Arrangements for Providing Improved Network Services to Wireless Handheld Devices," filed March 17, 2000 (Atty. Docket No. 1933.0020001, previously 95-345A), all of which are incorporated by reference herein in their entireties.

Cross-Reference to Related Applications

This patent application is potentially related to the following co-pending U.S. utility patent applications, which are herein incorporated by reference in their entireties:

"System, Method, and Computer Program Product for Server Side Processing in a Mobile Device Environment," Serial No. 09/705,914, Attorney Docket No. 1933.001000B filed on November 6, 2000.

Background of the Invention

Field of the Invention

The present invention relates generally to mobile communications, and more particularly relates to technology for using interactive applications while on-line and off-line on mobile devices.

Related Art

A variety of mobile devices (such as personal data assistants, or PDAs) exist. Such mobile devices include ones based on the Palm operating environment and the Windows CE operating environment.

A variety of software applications for those mobile devices also exist.

What does not exist prior to the invention are software applications for enabling web content (as well as other objects) to be loaded on mobile devices, and for users of mobile devices to operate with such web content on their mobile devices in an interactive manner while in an off-line mode.

Summary of the Invention

Briefly stated, the invention includes systems, methods, computer program products, and combinations and sub-combinations thereof for enabling web content (as well as other objects) to be loaded on mobile devices (as well as other types of devices), and for users of mobile devices to operate with such web content on their mobile devices in an interactive manner while in an off-line mode.

According to embodiments, the present invention performs script operations for mobile devices including steps for sending a request for an object

and a list of support languages, and receiving the object and any related scripts in the supported languages.

These and additional features and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters generally identify corresponding elements throughout.

Brief Description of the Figures

The accompanying drawings, which are incorporated herein and form part of the specification, illustrate embodiments of the present invention and, together with the description, further serve to explain the principles of embodiments of the invention.

FIG. 1A is a block diagram of an embodiment of the invention;

FIG. 1B is an alternative block diagram of the invention;

FIG. 1B1 is a block diagram of an example data processing unit useful in some embodiments for implementing items from FIGS. 1A and 1B;

FIGS. 1C, 1D, 1E, 1F, 1G, 1H, 1I, and 1J are used to generally describe embodiments of the invention;

FIG. 2A is a block diagram illustrating additional modules according to an embodiment of the invention;

FIG. 2B1 is a block diagram illustrating an additional module according to an embodiment of the invention;

FIG. 2B2 is a block diagram illustrating an additional module according to an embodiment of the invention;

FIG. 2C is a diagram illustrating some processing components according to an embodiment of the invention;

FIG. 2D1 is an example flowchart relating to structuring interactive content according to an embodiment of the invention;

FIG. 2D2 is an example flowchart relating to structuring and rendering interactive content according to another embodiment of the invention;

FIG. 2E is a diagram illustrating an example of content formatting according to an embodiment of the invention;

FIG. 2F1 is a diagram illustrating the content instantiation architecture according to an embodiment of the invention;

FIG. 2F2 is a diagram illustrating the content instantiation architecture according to another embodiment of the invention;

FIG. 2F3 is a flowchart relating to a optimized data modification according to an embodiment of the invention;

FIG. 2G is a diagram illustrating an example of a database architecture;

FIG. 2H is a diagram illustrating the content structure according to an embodiment of the invention;

FIGS. 2I1 and 2I2 demonstrate CSS style examples according to embodiments of the invention;

FIG. 2J demonstrates an example of floating images, where the text flows around the image;

FIG. 2K is an example architecture showing construction, layout, rendering, and cross-platform technology, according to embodiments of the invention;

FIG. 2L is an example operation where a device is able to sync with a current server, according to embodiments of the invention;

FIG. 3A is an example flowchart relating to a server cache for transformed content according to an embodiment of the invention;

FIG. 3B is an example flowchart relating to a server cache for transformed content having negative caching according to an embodiment of the invention;

FIG. 3C is an exemplary fetch diagram illustrating hits on a server occurring at the same time;

FIG. 3D is an example flow diagram representing a method for randomizing the expiration of objects set to expire at the same time according to an embodiment of the invention;

FIG. 3E is a diagram showing freshness lifetime for an object according to an embodiment of the present invention;

FIG. 3F is a block diagram illustrating a single account/profile having multiple devices according to an embodiment of the invention;

FIG. 3G shows an example screen shot for enabling a user to add multiple servers according to an embodiment of the invention;

FIG. 3H is an exemplary block diagram representing a single mobile device that connects to multiple servers according to an embodiment of the invention;

FIG. 3I is an exemplary flow diagram representing a sync process for a device connected to multiple servers according to an embodiment of the invention;

FIG. 3J is an exemplary diagram illustrating a multiple device – multiple server configuration according to an embodiment of the invention;

FIG. 4 is a block diagram illustrating script compiler modules according to an embodiment of the invention;

FIG. 5 is a block diagram illustrating a language interpreter module according to an embodiment of the invention;

FIG. 6 is a block diagram illustrating the operations of delegated language interpreter modules according to an embodiment of the invention;

FIG. 7 is a block diagram illustrating a server with a server-side language interpreter module according to an embodiment of the invention;

FIG. 8 is a flowchart showing a routine for script operations according to an embodiment of the invention;

FIG. 9 is a flowchart showing a routine for executing scripts according to an embodiment of the invention;

FIG. 10 is a flowchart showing a routine for updating properties according to an embodiment of the invention; and

FIG. 11 is a flowchart showing a routine for server-side script operations according to an embodiment of the invention.

It should be understood that these figures depict embodiments of the invention. Variations of these embodiments will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein. For example, the flow charts contained in these figures depict particular operational flows. However, the functions and steps contained in these flow charts can be performed in other sequences, as will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein.

Detailed Description of the Preferred Embodiments

1. Overview of Embodiments of the Present Invention

Embodiments of the present invention are briefly described in this section.

Briefly stated, the invention is directed to placing objects such as, but not limited to, Internet or Web content on data processing devices, such as but not limited to mobile devices. Table 1 lists examples of such Internet content, although the invention is not limited to these examples.

TABLE 1. Internet Content

Internet content includes but is not limited to:

HTML
JavaScript™
Channels
Java™
ActiveX
Multimedia:

Images (e.g., JPEG, GIF, PNG, vector graphics, etc.)
Audio Files (e.g. MP3)
Video (e.g. AVI)
Streaming Content: Voice/Data/Video
Binary files
XML
Applications
Data Objects
Documents
Anything that can be delivered via a "browser"

Table 2 lists examples of mobile devices, although the invention is not limited to these examples.

TABLE 2. Mobile Devices

Mobile devices include but are not limited to:

Handheld Computers
Cellular Phones
Internet-enabled Phones
Pagers
Radios
TVs
Audio Devices
Car Audio Systems
Recorders
Text-to-Speech Devices
Bar-code Scanners
Net Appliances
Mini-browsers
Personal Data Assistants (PDAs)

FIG. 1C illustrates the concept of the invention of placing objects on data processing devices, such as mobile devices.

1.1. Enabling Mobile Devices to Interact With Networked Applications

The invention includes technology for using applications on mobile devices that interact with the Internet or with intranets. The invention enables applications available via a network or via an Internet/intranet to download and to run on mobile devices. Consequently, the invention includes software and methods for administering a server that manages the variables relevant to a mobile device/server environment.

The invention enables:

Mobile devices to operate in conjunction with a Web server, even when the mobile devices are not coupled directly to the PC using portable on-device servers: Web pages are loaded, viewed, cached, and deleted even when the device is not coupled to any network.

Mobile devices to operate in conjunction with the Web, Internet, or intranet via a connection mechanism and then in disconnected mode or with the Web, Internet, or intranet in wireless mode with a continuous or a discontinuous connection mechanism.

A technique for interactive connectivity between handheld computers and computer networks.

Fleet management for centrally administering information in a handheld network environment that includes, but is not limited to, user data, user groups, group channels, channel data, personal channels, commercial channels, user accounts, corporate account, software groupings, personal information management, form delivery, form management, device configuration, device databases, device contents, and devices parameters.

Obtaining updated Web pages and other network objects, for use when the mobile device is not communicating with the PC.

An example mobile device/server environment is shown in FIG. 1D.

1.2. Rapid Transfer of Web Pages to Mobile Devices

To improve efficiency of data exchange between mobile devices and networked content, the invention includes an improved communication protocol that collects requests and responses for network objects into a smaller number of protocol (such as HTTP) requests and responses. The server also determines the nature and the resources of the mobile device. This protocol is represented, for example, in FIG. 1E.

Downstream, the data is encoded in a data format called content stream (tokenized version of the data) and sent to the device. The content stream format creates a tokenized codification of HTML pages that is sent to the device. (The device receives the content stream and presents the material on the device.)

The HTML page is encoded into the content stream and sent to the device. The encoding is a mapping of parent and child HTML elements and/or resources to alphanumeric values.

The sync operation of the invention includes various synchronization processes that can collect information from the Internet to a server, and to the client. In embodiments, the usage of the term "sync" refers to the overall operation of connecting a client to a server for the exchange, interaction, creation, and removal of data.

In one embodiment, syncing can be defined as mirroring data on a client and a server, such that the data is the same on client and server. In other embodiments, syncing can be defined as overwriting data on a client or on a server, such that the data on either a client replaces the data on a server, and vice versa.

In one embodiment, a sync operation involves a user placing a mobile device into an adapter that includes a sync button. The adapter is connected to a server. Upon pressing the sync button, the user initiates the sync operations of the present invention, which include various synchronization processes (specific delivery modes). Thus, the term sync is meant to refer to the overall operation

of linking a client to a server. Synchronization is meant to refer to the specific process of copying, adding, filtering, removing, updating and merging the information between a client and a server. Any number of synchronization processes can be executed during a sync.

Before being sent downstream the data is compared to the data that is known to be on the client and then the client is updated all at once in a one-up/one-down synchronization method, which is represented in FIG. 1F. The server sets the client to preemptively prepare all device information necessary during the sync. Then the server receives the set of information in a one-up fashion. The server collates the information and sends the information in a one-down fashion. This optimizes the sync's efficiency and speed. The sync process, according to embodiments of the invention, is represented in FIGS. 1G and 1H.

1.3. Optimizing Content of Web Pages for Mobile Devices

When Web content and other network objects pass through the server they are processed to minimize their size and to optimize their delivery to mobile devices: for presentation, for ease of use, for efficiency, for size, etc.

The invention uses server logic to optimize content. The server assesses the mobile device to optimize web content for the particular device. Factors that the server logic considers when performing this optimization include, but are not limited to (it is noted that the server may consider subsets of the following, depending on the application and implementation):

Dynamic memory specifications

High memory specifications

Protected Memory

Storage Memory

Database Memory

Available storage space

Screen size

User profile(s)

Color depth

Applications on device

Buttons on-device

Data markers (e.g., cookies, tokens)

Preferences

Fonts

Font specifications

Sync type

Synchronization types

Supported data types

Supported mime types

Connection/Network profile

An example optimization process is shown in FIG. 1I.

On the server, the graphic is optimized per the state information of the device. If the device sends down the need for the graphic on a page for a device with a display that is 27 cm wide and in grayscale, the server sends its best version of a graphic optimized for that environment.

The technology of the invention is extended by tags on HTML pages that identify content that is designed for additional modifications. Any and all bytes processed by the server are potentially examined for compression/optimization. The server detects the tag and executes the necessary logic.

Table 3 illustrates example tags (the invention is not limited to the tags shown in Table 3).

TABLE 3. Sample Markup Language

Tag	Effect
<META NAME="Handheld-Friendly" content="True">	This tag enables several HTML features that are normally turned off. Most notably, The invention does not try to display

	TABLE tags or the HSPACE and VSPACE attributes of IMG tags unless the page is marked as "HandheldFriendly". Most TABLEs or H/VSPACEs are designed for much larger screens.
<AGIGNORE></AGIGNORE>	Used in a wireless channel. Use the AGIGNORE tag to surround content within an HTML page that may be inappropriate or unattractive on Internet-enabled phones.
<AGPAGEBREAK TITLE="your title">	Used in a wireless channel. Breaks up pages on request. When processing pages for devices other than WAP phones, the server ignores the AGPAGEBREAK tag.

Web Content Aggregation, Web Channel Development, and Web Content Delivery for Users of the Internet and of Mobile Devices

The invention is extended by the coupling of devices to the content available at the server web site (see the example shown in FIG. 1J).

These and other embodiments of the present invention are described in greater detail below.

Structural Embodiments of the Present Invention

FIG. 1A is a block diagram of a data processing environment 102 according to an embodiment of the invention. The data processing environment 102 includes a server 104 (although only one server 104 is shown, in practice the data processing environment 102 may include a plurality of servers), one or more devices 106, one or more adapters 118, and one or more providers 128.

Generally, the server 104 maintains a collection of channels. In an embodiment, a channel comprises a collection of objects. An object is any entity that can be transferred to a client 108, such as but not limited to content, applications, services, images, movies, music, links, etc.

A channel includes a number of properties. At least some of these properties define the objects that the channel includes. Such properties include, but are not limited to, the following (properties of channels may vary depending on the application and/or implementation):

A name of the channel.

A location of a root object (such as but not limited to a URL). In an embodiment, this root object is included in the channel. An indication of the number of levels below the root object, for which to include objects in the channel. For example, in an embodiment, if this property is equal to "1 level," then all objects that are 1 level down from the root object (reached by traversing links in the root object), are included in the channel. If this property is equal to "2 levels," then all objects that are 1 level down from the root object (reached by traversing links in the root object), and all objects that are 1 level down from those objects (reached by traversing links in those objects), are included in the channel. Embodiments of the invention allow "uneven" trees, where some branches of the tree extent to a greater number of levels than other branches of the tree. In other embodiments, the trees are even or balanced.

A maximum size of the channel. For example, if this is set to 500 Kbytes, then the aggregate size of the objects in the channel cannot be greater than 500 Kbytes. If the aggregate size of the objects in the channel is greater than this value, then embodiments of the invention may delete objects from the channel and/or delete portions of objects in the channel.

An indication of which resource objects are enabled for the channel:

An indication of whether or not images are to be included in or excluded from objects in the channel; and

An indication of whether or not scripts are enabled in objects in the channel.

A refresh methodology.

It is noted that the properties associated with channels may vary from implementation to implementation. Also, implementations may employ

combinations of the above properties, and/or properties in addition to the following, as will be appreciated by persons skilled in the relevant art(s).

The invention includes processes for managing channels, including but not limited to adding channels to the collection of channels maintained by the server 104.

The server 104 offers channels to clients 108. A user associated with or on behalf of a client 108 may access the server 104 and view the collection of channels. The client 108 (via the user, for example) may then select any combination of the channels in the collection. The server 104 maintains a list of the channels associated with each of the clients 108.

During a synchronization process, the server 104 loads a device 108 with the channels associated with the client 108. Generally, the server 104 does this by obtaining from providers 128 the objects defined by the channels, and causing those objects to be stored on the client 108. Thus, during the synchronization process, the server 104 will load the client 108 with the selected channels. More particularly, the server 104 will load the client 108 with the objects associated with the channels.

The client 108 may process and use those objects when not connected to the server 104. The invention enables the client 108 to actively interact with the objects and channels.

In one embodiment, the client 108A directly interacts with the server 104 via some transmission medium 120B, which may be any wired or wireless medium using any communication protocol.

In another embodiment, the client 108B indirectly interacts with the server 104 via an adapter 118. For example, the client 108B may be a mobile device (such as a Palm device) and the adapter 118 may be a cradle and a computer coupled to the cradle (the mobile device is inserted into the cradle). In this instance, the adapter 118 presents itself to the server 104 as a client 108B (via client communications module 110C). When the server 104 sends objects to the adapter 118, the adapter interface module 116 writes those objects to client 108B.

In embodiments, adapter interface module 116 can be a Hot Sync™ Manager, an Active Sync™, etc. It is noted that the invention is not limited to any of the implementation examples discussed herein.

The components shown in FIG. 1A shall now be described in greater detail.

The server 104 includes an administration module 122, a database module 126, a user interface 130, a web synchronization module 124, a server extension module 156, a fleet management module 154, a notification module 132, and a server communication module 114. Other embodiments of server 104 may include a subset of these modules, and/or may include additional modules.

The administration module 122 controls and manages the states of the server 104 and the clients 108. For example, the administration module 122 manages and controls groups of clients 108, permissions assigned to clients 108, groups, and channels. For example, the administration module 122 administers the users/clients 108 assigned to groups, and the channels associated with users. These and additional functions performed by the administration module 122 are described herein.

The database module 126 controls access to databases associated with the server 104. The database module 126 maintains information relevant to the clients 108, as well as information relevant to the modules contained in the server 104. The database module 126 manages information on the collection of channels maintained by server 104. These and additional functions performed by the database module 126 are described herein.

The user interface 130 is, in an embodiment, a graphical user interface (GUI) that enables users and clients 108 to access functions and modules offered by the server 104. More generally, the user interface 130 within server 104 provides access to server 104 and the modules and resources contained therein.

The invention supports various server web sites that are available through any communication medium, such as but not limited to the Internet, intranets, direct dial up links, etc. The UI 130 enables such web sites.

These and additional functions performed by the user interface 130 are described herein.

The web synchronization module 124 is an application/instance of server extension module 156, and controls synchronization of web content to client 108. The invention may include other synchronization modules (which are application/ instances of server extension module 156) that control synchronization of other types of objects to clients 108. For example, the server 104 may administer a calendar that may be installed on clients 108. The synchronization of appointments, events and/or dates on this calendar between clients 108 and the server 104 may be performed by a calendar synchronization module. These and additional functions performed by the server extension module 156 are described herein.

The fleet management module 154 performs functions associated with fleets of clients 108, which are groups of clients 108. For example, fleet management module 154 may perform global or mass operations on groups (fleets) of clients 108, such as loading or updating an application on groups (fleets) of clients 108. Another example of a mass operation is retrieval of information on clients 108 in a fleet, such as the free memory in clients 108 in a fleet (this would help an organization determine if its clients 108 need a memory upgrade). These and additional functions performed by the fleet management module 154 are described herein.

The server extension interface/module 156 enables modules, such as third party modules, to operate in or work with the server 104 (and modules contained in the server 104). The server extension module 156 presents an API (application programming interface). Modules in the server 104 may operate with other devices in the server 104 by conforming to the server API.

For example, the web synchronization module 124 and the fleet management module 154 (as well as other types of synchronization modules, not shown in FIG. 1A) may interact with databases on the server 104 via the database module 126 by going through the server extension module 156. The web

synchronization module 124 and the fleet management module 154 may not be able to interact directly with the database module 126 for a number of reasons. For example, they may support different data formats, or simply "speak different languages." However, they can interact via the server extension module 156 as well as other server modules as long as they conform to the API of the server extension module 156. This is true of any modules in the server 104, or that interact with the server 104.

Server communication module 114 enables communication between the server 104 and entities external to the server 104, such as clients 108, adapters 118, providers 128, work stations, etc. The server 104 communicates with these entities via communication mediums 120, which may be any type of wireless or wired communication using any protocol. It is noted that multiple server communication modules 114 may execute in a single server 104. For example, in one embodiment, server communication module 114 is a TCP/IP stack. In another embodiment, server communication module 114 is a secure socket layer stack or a compression stack. The invention is not limited to any implementation examples discussed herein. These and additional functions performed by the server communication module 114 are described herein.

The notification module 132 sends objects to clients 108 beyond objects related to channels associated with clients 108. Such objects could be requested by client 108 in advance. For example, a client 108 could ask for a notification when an event happens, such as when a stock reaches a target price. When the event occurs, the notification module 132 would cause an appropriate notification(s)/object(s) to be sent to the client 108. Alternatively, the notification module 132 may send objects to clients 108 without any prior explicit request from the client 108. For example, the notification module 132 might send channels to clients 108 when such channels are identified to be similar to those already selected by the clients 108. Also, the notification module 132 might send appropriate notifications/objects to the clients 108 when such clients 108 receive email or faxes at the server 104. In embodiments, the notification module

132 transmits such objects to the client 108 immediately when the event occurs, during the next synchronization with the client 108, or at some other future synchronization.

An alternative representation of server 104 is shown in FIG. 1B. FIG. 1B illustrates, for example, that messages from entities outside of server 104 are received by server extension interface/module 156 via server communications modules 114. Generally, such messages represent requests for the server 104 to perform various functions. The server extension module 156 conceptually operates as a dispatcher who routes such messages to other modules contained in the server 104, such as web synchronization module 124 (who handles requests to synchronize with web content), notification module 132, fleet management module 154 (who handles fleet related requests), and/or third party modules 155 (such as other synchronization modules). Thus, the invention supports modules 155 generated by third parties to perform various functions. Such modules 155 "plug-in" to the server 104 via the server extension module 156.

Referring again to FIG. 1A, the devices 106 may be any type of data processing device. In embodiments of the invention, the devices 106 are mobile computing devices, although the invention is not limited to these embodiments. In such example embodiments, the devices 106 may include, but are not limited to, handheld computers, cellular phones, internet-enabled phones, pagers, radios, tvs, audio devices, car audio systems, recorders, text-to-speech devices, bar-code scanners, net appliances, mini-browsers, personal data assistants (PDAs), etc.

In embodiments of the invention, the devices 106 include software, hardware, and/or combinations thereof related to client functionality (such client functionality is described herein). When a device 106 includes such software, hardware, and/or combinations thereof, the device 106 is referred to herein as a client 108. Accordingly, it can be said that the data processing environment 102 includes one or more clients 108.

Clients 108 each may include a layout and rendering module 134, a forms module 136, a control module 142, a user interface 144, a client extension

interface 138, a client interface module 112, a client communications module 110, a JavaScript™ engine 140, and a database module 146. Other embodiments of clients 108 may include a subset of these modules, and/or may include additional modules.

Layout and rendering module 134 controls the processing of data objects on client 108, such as the layout and rendering of data objects on client 108. For example, the layout portion of module 134 obtains information from databases of the client 108 (via the database manager 146) and determines where such information should be rendered on the display of the client 108. Such information may include anything that can be rendered, such as but not limited to images, text, links, etc. The rendering portion of module 134 is responsible for drawing items on the display (drawing bits to the screen). These and additional functions performed by the layout and rendering module 134 are described herein.

The forms module 136 controls and manages forms. For example, in embodiments the forms module 136 manages aspects of off-line forms, such as HTML forms and/or multi-page forms. The forms module 136 enables access to and user interaction with forms (in some embodiments, the forms module 136 via UI 144 enables users of client 108 to directly access forms). The forms module 136 maintains the status of forms. Forms module 136 can also include a forms manager (not shown) to provide added functionality. These and additional functions performed by the forms module 136 are described herein.

The user interface 144 is preferably a graphical user interface that enables users to interact with client 108 and functions and modules provided by the client 108. More generally, UI 144 controls how functions presented by modules of the client 108 are presented to users. The UI 144 controls how users interact with such functions and modules. It is noted that the functionality of the UI 144 may be distributed. For example, portions of the UI 144 may reside in the forms module 136, as well as other modules of client 108. These and additional functions performed by the user interface 144 are described herein.

The client extension interface 138 enables modules, such as third party modules, to operate in or work with the client 108 (and modules contained in the client 108). The client extension interface 138, also known as an on-device server, presents an API (application programming interface) that is, in embodiments, common to clients 108 on many architectures.

Modules in the client 108 can work together via the client extension interface 138. For example, the JavaScript™ engine 140 may decide that it wishes to display a message to the user. To do this, the JavaScript™ engine 140 would work through the client extension interface 138 to cause the UI 144 to display the message to the user. The JavaScript™ engine 140 may not know how to directly interact with the UI 144. However, as long as both the JavaScript™ engine 140 and the UI 144 conform to the API of the client extension interface 138, then they can operate together.

Similarly, the control module 142 may decide that it needs to store some data in a database. The control module 142 would do this by working with the client extension interface 138 to access the database module 146 to effect such a modification to the databases in the client 108. These and additional functions performed by the client extension interface 138 are described herein.

The JavaScript™ engine 140 executes objects written in the JavaScript™ language that operate on client 108. As noted, the JavaScript™ engine 140 conforms to the API of the client extension interface 138, and works with the client extension interface 138 to work with other modules in client 108. These and additional functions performed by the JavaScript™ engine 140 are described herein.

Although not shown in FIG. 1A, embodiments of the invention include other engines for executing other types of scripts on client 108. These other engines can interact with other modules on client 108 as long as the engines conform to the API of the client extension interface 138.

The database module 146 controls access to databases associated with client 108. More generally, the database manager 146 controls access to

resources on the client 108. For example, the control module 142 may interact with the database manager 146 to open an address book in the databases, and to write a record to the address book. Alternatively, the forms module 136 can interact with the database module 146 to access forms that are stored in the databases. These and additional functions performed by the database module 146 are described herein.

Client communications module 110 enables the client 108 to interact with external entities, such as server 104. In embodiments, the client communications module 110 enables TCP/IP traffic, although the invention is not limited to this example. More generally, the client communications module 110 enables communication over any type of communication medium 120, such as wireless, wired, etc., using any communication protocol, such as a pager protocol. These and additional functions performed by the client communications module 110 are described herein. The client interface module 112 enables the client 108 to communicate with adapters 118. Client interface module 112 optionally links to client communications module 110 in some embodiments to provide functionality (for example, when the client communications module 110 uses a wireless modem's drivers, which are accessed via client interface module 112). In embodiments, the client interface module 112 may be Hot Sync™ Manager in the Palm operating environment, or Active Sync™ in the Windows CE™ operating environment, or Pilot Link™ in the Unix operating environment. It is noted that these implementation examples are provided for illustrative purposes only. The invention is not limited to these examples. These and additional functions performed by the client interface module 112 are described herein.

The control module 142 coordinates the activities of the other modules in client 108 so that all the modules share resources properly. For instance, control module 142 can determine priorities for shared resources such as processing time, accessing memory, etc.

Providers 128 are sources of various types of objects, such as but not limited to content (content providers 128A), applications (application providers

128B), services (service providers 128C), etc. Providers 128 may also include servers 104' (similar to server 104), which may provide objects such as but not limited to content, applications, services, etc. For example, and without limitation, the application providers 128B may provide objects relating to (without limitation) operating system updates/changes, system upgrades, application updates/changes, etc.

Adapters 118 include an adapter interface module 116, a user interface 148, a database module 150, an adapter synchronization module 152, and a client communications module 110. Other embodiments of adapters 118 may include a subset of these modules, and/or may include additional modules.

Client communications module 110 is the same as similarly named modules in clients 108.

The adapter interface module 116 enables the adapter 118 to communicate with clients 108.

The adapter synchronization module 152 is involved with synchronization operations between server 104 and clients 108.

The UI 148 enables users to interact with modules and functions of adapter 118.

The database module 150 controls access to databases associated with adapter 118. The database module 150 manages information needed for clients 108 to remain in sync with server 104. In some embodiments, the adapter 118 does not include the database module 150 or the UI 148 (i.e., in embodiments where the adapter 118 operates essentially as a pipe, as in some embodiments on Unix).

These and additional functions performed by modules of the adapter 118 are described herein.

Additional modules and features of embodiments of the invention are described below.

1.4 Example Implementation Embodiments

FIG. 1B1 illustrates a block diagram of a data processing unit 103A that can be used to implement the entities shown in FIGS. 1A and 1B. It is noted that the entities shown in FIGS. 1A and 1B may be implemented using any number of data processing units 103A, and the configuration actually used is implementation specific.

Data processing unit 103A may represent laptop computers, hand held computers, lap top computers, and/or any other type of data processing devices. Which type of data processing device used to implement entities shown in FIGS. 1A and 1B is implementation specific.

Data processing unit 103A includes a communication medium 103B (such as a bus, for example) to which other modules are attached.

Data processing unit 103A includes one or more processor(s) 103C, and a main memory 103D. Main memory 103D may be RAM, ROM, or any other memory type, or combinations thereof.

Data processing unit 103A may include secondary storage devices 103E, such as but not limited to hard drives 103F or computer program product interfaces 103G. Computer program product interfaces 103G are devices that access objects (such as information and/or software) stored in computer program products 103. Examples of computer program product interfaces 103G include, but are not limited to, floppy drives, ZIP™ drives, JAZ™ drives, optical storage devices, etc. Examples of computer program products 103H include, but are not limited to, floppy disks, ZIP™ and JAZ™ disks, memory sticks, memory cards, or any other medium on which objects may be stored.

The computer program products 103H include computer useable mediums in which objects may be stored, such as but not limited to optical mediums, magnetic mediums, etc.

Control logic or software may be stored in main memory 103D, secondary storage device(s) 103E, and/or computer program products 103H.

More generally, the term "computer program product" refers to any device in which control logic (software) is stored, so in this context a computer program product could be any memory device having control logic stored therein. The invention is directed to computer program products having stored therein software that enables a computer/processor to perform functions of the invention as described herein.

The data processing unit 103A may also include an interface 103J which may receive objects (such as data, applications, software, images, etc.) from external entities 103N via any communication mediums including wired and wireless communication mediums. In such cases, the objects 103L are transported between external entities 103N and interface 103J via signals 103K, 103M. In other words, such signals 103K, 103M include or represent control logic for enabling a processor or computer to perform functions of the invention. According to embodiments of the invention, such signals 103K, 103M are also considered to be computer program products, and the invention is directed to such computer program products.

2. *A Cross-Platform Browser and Cleint/Server Software Innovation for Mobile Devices*

As described above, the technology uses a cross-platform strategy for serving and obtaining content requests on and across platforms and processors available to mobile devices. In some embodiments, the client enables desktop personal computer (PC) functionality on mobile devices. For example, the client can support dynamic hypertext mark-up language (DHTML) on mobile devices; it can support device-side interpretation of JavaScript™; and it can provide secure client/secure protocol/secure server interaction. It is noted that these examples are mentioned for illustrative purposes only, and are not limiting. Addition functions and capabilities are within the scope and spirit of the present invention, as will be appreciated by persons skilled in the relevant art(s) based on the teachings contained herein.

Additionally, the client 108 of the invention enables per channel (and/or per page) username and password authentication for transactions (e.g. in e-commerce applications and/or channels), digital notarization, content hold and deliver technology in connected and disconnected modes, and bookmarks. Furthermore, the clients 108 (i.e., client 108A and 108B) of the invention enable support for multiple protocols such as mailto and dialto interfaces, and DHTML. It is noted that these examples are mentioned for illustrative purposes only, and are not limiting. The invention is applicable to other protocols, as will be appreciated by persons skilled in the relevant art(s) based on the teachings contained herein.

In an embodiment, the client 108 of the invention integrates with other mobile device applications through methods such as but not limited to: cut and paste, domain integration of Find and/or Search methods, and mobile device communication between on-device applications and their separate tables of data. For example, the client 108 of the invention can invoke a vector graphics display or a word processor or spreadsheet file synced to the device. In one embodiment, these features correspond and extend the functionality of pluggable MIME types on server 104.

In embodiments, the client 108 is designed to support the additional Internet document standards: HTML 4.0, XHTML 1.0, CSS (Cascading Style Sheets), and the W3C DOM (Document Object Model). It is noted that these examples are mentioned for illustrative purposes only, and are not limiting. The invention is applicable to other standards, as will be appreciated by persons skilled in the relevant art(s) based on the teachings contained herein.

Referring to FIG. 2A, a block diagram, illustrating additional modules according to an embodiment of the invention, is shown.

Server 104 can include parser module 201A, layout module 201B, and/or proxy rendering module 201C. Modules 201A – 201C can be implemented in server 104 alone or in combination with other elements or modules, such as in combination with clients (such functionality can also be performed completely

by clients). It is noted that the functionality associated with modules may vary from implementation to implementation. The specific functionality and implementation described herein represent an example embodiment of the invention. Other embodiments will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein.

Also, implementations may employ combinations of the above modules, and/or employ the functionalities of the above modules as sub-modules to other modules of server 104, as will be appreciated by persons skilled in the relevant art(s).

In one embodiment, parser module 201A reads the objects on a page, such as a Web page. The parser module 201A separates out the description of each object on the page and generates a tree of objects based on their descriptions and the inherent relationship defined by the descriptions. In one embodiment, the tree of objects is compatible with the W3C DOM. Thus, in the case of HTML, the Web page is a description of content objects using tags, attributes, and styles. In the case of WML, the page is a description of content object using binary data.

In one embodiment, layout module 201B maps the parsed objects of a page and determines how the objects can be positioned and sized (or laid out) in order to provide a page with similar substance on the mobile device to which it is going to be transmitted. As with the other embodiments of the layout module described herein, the layout module 201B receives display, font color, and other device configuration information from the proxy render module 201C.

In a server-side layout embodiment, layout module 201B receives the display configuration of the mobile device from proxy render module 201C. Proxy render module 201C provides server 104 with the functionality of layout and rendering module 134 (in this case, either of modules 134A or 134B) for any number of clients 108. In one embodiment, the proxy render module 201C provides configuration information from a database, which stores previously obtained configuration information. In embodiments described herein, the server 104 receives the configuration information from the client 108. For example,

server 104 receives the configuration information from layout and rendering module 134. From this information, server 104 is able to operate proxy render module 201C accordingly and provide data in the proper format. For example, in one embodiment, the proxy render module 201C is able to produce byte code in the form of a content stream compatible with the display hardware of device 106.

As such, proxy rendering module 201C determines the display capabilities of the device 106 to which the content stream is going to be transmitted. Furthermore, in one embodiment, proxy rendering module 201C determines information about how particular objects can appear (i.e., are positioned) on the display of a mobile device from device information stored in database module 126, rather than for a specific device 106 from information provided by client 108.

Furthermore, the functionality of these modules can be implemented on clients 108. For example, the rendering functionality of layout and rendering module 134 can provide the configuration information directly to the layout functionality of the same module. In embodiments, modules implemented on the client 108 are designed to operate on relatively small computers and/or mobile devices such as those described herein as well as those similarly designed. The combination of functionalities in one module 134 is for illustrative purposes. The combination may be separated and implemented in two separate modules as discussed with respect to server 104 above. These implementations are discussed in more detail below.

Referring to FIG. 2B1, a block diagram, illustrating an additional module according to an embodiment of the invention, is shown.

Client 108A can include client parser module 202A. Client parser module 202A provides functionality similar to parser module 201A as described herein.

Referring to FIG. 2B2, a block diagram, illustrating additional modules according to an embodiment of the invention, is shown.

Client 108B can include client parser module 202B and client interface module 112B. Client parser module 202B provides functionality similar to parser module 201A as described herein. Client interface module 112B is capable of connecting with providers 128 via communication medium 120E. Providers 128, as described herein, are content providers, such as Web sites. Communications medium 120E may be implemented to augment the connection with server 104 via adapter 118. For example, communications medium 120E, as shown, could connect directly to providers 128 via a wireless link in order to obtain updated content from providers 128 or to transmit data to providers 128.

In one embodiment, the parsing functionality added to client 108 via client parser modules 202A and 202B allows clients 108 to obtain content directly from providers 128 without server 104 transformation. Some embodiments of this distribution of functionality is discussed below with respect to FIG. 2C.

Referring to FIG. 2C, a diagram, illustrating some processing components according to embodiments of the invention, is shown.

Content 203A from a page in the form of objects is parsed by parser module 203B, laid out by layout module 203C, rendered by rendering module 203D, and sent to mobile devices for display in the form of pixel data 203E. These objects can include but are not limited to tags, attributes, and style information.

The modules 203B, 203C, and 203D are similar to the modules discussed in FIGS. 2A, 2B1, and 2B2. FIG 2C illustrates the feature of the invention where the method, system and computer program product can be configured such that the operations of modules can be performed on server 104 and/or clients 108, as well as on adapter 118 as well as combinations thereof.

In the examples of FIG. 2C, the operational processes of each of modules 203B, 203C, and 203D are delineated by marker 203F for parser module 203B. Similarly, marker 203G delineates the operational processes of layout module 203C. Marker 203H shows the operational processes for rendering module 203D.

In embodiments of the invention, the operational processes of these modules, as well as the modules themselves, can be implemented on either or both the server 104 or the client 108 (or adapter 118). This flexibility allows the invention to optimize the transformation and delivery of content to mobile devices based on the capabilities of the devices and/or requirements of users.

Line 203I illustrates an example implementation where modules 203B and 203C are operating on server 104 to perform the functions of parsing and layout. The asterisk of line 203I shows the transition to client 108 (or adapter 118) and thus the operations of module 203D are performed on client 108.

Similarly, lines 203J, 203K, and 203L illustrate other possible apportioning of the operational tasks. The markers between the arrows of lines 203J, 203K, and 203L similarly showing the transition from one of the server 104 to client 108 (and/or adapter 118).

In additional embodiments, modules on both server 104 and client 108 (and adapter 118) can operate in parallel or series on the objects of content 203A.

Referring to FIG. 2D1, an example flowchart 204 relating to structuring interactive content, according to an embodiment of the invention, is shown.

In step 204A, server 104 receives a request for pages. In one embodiment, the client 108 sends the request.

In step 204B, server 104 receives mobile device and client information describing the capabilities of the client 108 and the device 106. In one embodiment, client 108 sends information regarding the display and memory specifications of the mobile device upon which it is operating.

In step 204C, server 104 parses the pages into a mutable document of content according to the device and client information of step 204B. In one embodiment, parser module 201A parses the pages into discrete objects.

In step 204D, server 104 determines the rendering parameters of the client and mobile device according to the information obtained in step 204B. In one embodiment, proxy rendering module 201C provides the rendering parameters of the client and mobile device.

In step 204E, server 104 lays out the document content according to the rendering parameters determined in step 204D. In one embodiment, the parsed objects of a page are assembled and formatted such that the page displayed by the client on the mobile device has the same functional display or presentation as on any other device. In an embodiment, layout module 201B provides common layout services for the server 104. Similarly, layout and rendering module 134A of FIG. 1A provides these services on client 108A.

In step 204F, server 104 determines the document table and document content to be sent to client 108 so that the client 108 can use the content of the page(s). The structure and format of the document table and document content (according to embodiments) are discussed below.

In step 204G, server 104 compresses the document content, preferably on a discrete object-by-object basis (although other embodiments are possible). For example, the discrete objects obtained from the parsing of step 204C are compressed individually in step 204G.

In step 204H, server 104 encrypts the document content, preferably on a discrete object-by-object basis. For example, the discrete objects obtained from the parsing of step 204C are encrypted individually in step 204H.

In step 204I, server 104 serializes the document content according to the discrete basis. For example, the document content is placed in ordered blocks or sections. In one embodiment, the discrete basis may prioritize index or home pages and place them in places in the serialized chain of objects so that they may be readily recalled.

In step 204J, server 104 serializes the document attributes related to the document content according to the discrete basis. In one embodiment, the document attributes are placed in a similar order as the document objects in step 204I. In another embodiment, the document attributes are serialized based on the type of objects they respectively identify.

In step 204K, server 104 transmits the serialized document to client 108A and/or adapter 118 for delivery to client 108B. In one embodiment, the serialized

document is a content stream transmitted to the device for client 108. The serialized document can provide an optimized format for delivering content to mobile devices that may not be designed to accommodate the relatively large file formats, which PCs are used to handling.

In the above description, server 104 is discussed as performing the operations of flowcharts 204. This is just one embodiment of the invention. Variations of this embodiment will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein. For example, client 108 can perform some or all of the operations of flowchart 204, as discussed below.

Referring to FIG. 2D2, an example flowchart 204M relating to structuring interactive content according to another embodiment of the invention is shown.

In step 204N, client 108 (as discussed above, client 108 is used to refer generically to client 108A and/or client 108B, unless stated otherwise) sends a request for pages.

In step 204Q, client 108 receives the requested pages. In one embodiment, client 108B receives the pages directly from providers 128, as shown in FIG. 2B2. In another embodiment, client 108 receives the requested pages from the server 104.

In step 204R, client 108 parses the pages into a mutable document of content. As described above, the mutable document allows for better access and storage for the client.

In step 204S, client 108 determines the rendering parameters according to the local render module, such as layout and rendering module 134.

In step 204T, client 108 lays out the document of content according to the rendering parameters determined by the local render module, such as layout and rendering module 134. Such parameters include, for example, colors supported, device screen size, font characteristics, etc.

In step 204U, client 108 displays the data on the screen. In one embodiment, the render module of client 108 puts the pixel data on the screen.

For example, the client communications module 110A receives the pages from server 104. Client parser module 202A parses the pages. Layout and rendering module 134A determines the rendering and layout parameters and forwards the pixel data to the screen. Other examples and embodiments are discussed in detail below.

2.1 Serialized Document Model

Similar to the PODS described herein, the document (data object) assembled through the steps of flowcharts 204 and 204M includes a structure that is advantageous for mobile devices. The document object models (DOM) of an embodiment of the invention supports Copy-On-Write which is usually only implemented in hardware. This enables the creation of discretely compressed read-only documents (for example, the pre-processed content stream as described herein) that can be modified and saved and then loaded and modified incrementally on a discrete basis.

This embodiment allocates data in its final form. In an embodiment, read-only data in the content stream is, as best can be achieved, in its final, usable form. The data does not have to be duplicated for use as with other types of models. This aspect of the invention is important given the relatively limited resources of mobile devices.

2.2 Memory Allocation

Furthermore, this model has a tight memory allocation scheme. In an embodiment, memory is allocated in Pools and Arenas that are scoped to the lifetime of the object being allocated. For example, documents have a memory scope. When an object whose lifetime is that of a document is allocated, it is allocated from the owner document's memory scope.

When the document is destroyed, the associated memory scope can be freed immediately. Objects that will exist for the life of a memory scope will be allocated from the scope Arena (in an embodiment, arena allocated objects cannot be individually freed). In an embodiment, objects whose lifetimes are more volatile will be allocated from the scope pool (where they can be later freed). This allocation is but an example and others can be implemented depending on the device's memory structure (and also possibly depending on other factors) as one skilled in the relevant art would recognize. As in the above example, when both arena and pool memory types exist, content can be stored in a read-only (arena) and/or in a writeable (pool). Further embodiments using this type of memory allocation are discussed below.

Referring to FIG. 2E, a diagram illustrating an example of content formatting according to an embodiment of the invention is shown. In this embodiment, an example button is shown within the document 205A. The button object includes a header 205B and the parameter information stored as data 205C. Header 205B can include type information which provides a mechanism whereby there is a pointer or pointers to the type of functions which operate on the data 205C.

Referring to FIG. 2G, a diagram illustrating an object model related to those shown in FIGS. 2F1 and 2F2 is shown. FIG. 2G shows a button object 207A that includes object pointers 207K. Object pointers 207K comprise a vtable pointer 207B and a data object 207C. Vtable pointer 207B points to a vtable 207D that contains function pointers 207F for accessing instance methods 207H.

Button object 207A and data 207C can be placed in writeable memory. Qualitative data can be read and written (thus, modified) by instance methods 207H, which are designed to read and manipulate data 207C.

There are drawbacks to this object model. They include, but are not limited to:

1) Data 207C is in writeable memory. Data 207C is relatively large. If data 207C is first available in read-only form, then it must first be copied into writeable memory.

2) Writeable memory is often scarce on mobile devices (as well as other devices).

3) Data 207C cannot easily be maintained in compressed form as the size of data 207C will change as it is modified. This requires additional writeable memory.

Additionally, the lack of a document table effectively means that any changes to the data 207C or the type information in header 207B requires a change to the vtable 207D which is used to operate on data 207C. These changes have to be made globally so that the entire document is recreated.

FIGS. 2F1 and 2F2 illustrate content instantiation architectures according to embodiments of the invention.

Referring to FIG. 2F2, a diagram illustrating the content instantiation architecture according to an embodiment of the invention is shown. FIG 2F2 differs from FIG. 2G in that data 206K2 is separated from object 206A. The addition of attribute pointer 206C, which points to data 206K2, serves as a link so that changes do not have to be made globally as in FIG. 2G.

Referring to FIG. 2F1, a diagram illustrating the content instantiation architecture according to an embodiment of the invention is shown. FIG. 2F1 is differs from FIG. 2F2 in that data 206K can be read only and compressed. Instance methods 206H are designed to read the compressed data.

The architectures of FIGS. 2F1 and 2F2 have advantages over the architecture of FIG. 2G. Unlike the architecture of FIG. 2G, data does not have to be decompressed or copied to writeable memory for initial use. Data can be read, and/or displayed immediately. The architectures of FIGS. 2F1 and 2F2 are very efficient for mobile devices (as well as other devices), which can have relatively limited memory or processing capabilities, because they use compressed read-only data. Additionally, the architectures of FIGS. 2F1 and 2F2

are useful for applications where memory and processing requirements need to be optimized, as well as for other applications as will be appreciated by persons skilled in the relevant art(s) based on the teachings contained herein.

Referring to FIG. 2F1, for example, document table 206A provides two pointers: Vtable pointer 206B and attribute pointer 206C. Vtable pointer 206B points to a vtable 206D which includes header 206F and function pointers 206G. Header 206F includes information pertaining to the class and type of functions being called. Function pointers 206G includes global, user, and low-level function pointers that provide access to the instance methods 206H of functions 206E.

Attribute pointer 206C points to the specific object in the content stream 206I similar to that discussed with respect to FIG. 2E. Header 206J and data 206K provide specific information about how to perform the functions pointed to in the vtable.

The use of document table 206A allows the entire document to be incrementally altered as may be required without having to totally restructure the document.

It is noted that the data 206K can be compressed. Content stream 206I, header 206J, and data 206K can be in read-only memory. Instance methods 206H can be designed to interpret the compressed data 206K so that instance methods 206H can read, display and process data 206K properly.

According to embodiments of the invention, modifications to data 206K are possible, but any modified data 206K objects are stored in writeable memory. Attribute pointer 206C is updated to reflect the modification so that future use of the specific data 206K object is directed to the modified object. The embodiments of FIGS. 2F1 and 2F2 therefore provide for relatively less use of writeable memory on the mobile device.

Software Modification on Write Method and Example

The embodiments of FIGS. 2F1 and 2F2 offer important advantages to browsers. For example, mobile devices typically are designed around several different operating systems and/or hardware standards. The use of a client that enhances access helps to assure compatibility. Furthermore, the general features of mobile devices are relatively limited compared to their PC counterparts. As such, mobile devices benefit from enhanced content storage, retrieval and modification techniques.

In an embodiment, data can be transformed as shown in FIGS. 2F1 and 2F2. In an example embodiment, the objects 206A in FIG. 2F1 and 2F2 share the same form, i.e., a compressed, read-only form. The read-only objects 206A of FIG. 2F1 can be transformed by instance method 206H into the writeable form shown in FIG. 2F2. In one embodiment, when instance method 206H is called, a software exception similar to a hardware exception occurs.

Referring to FIG. 2F3, a flowchart 206P, relating to an enhanced data modification process according to an embodiment of the invention, is shown. For illustrative purposes, FIG. 2F3 is described with reference to FIG. 2F1. However, the operations of FIG. 2F3 are applicable to other embodiments, such as the example embodiment of FIG. 2F2.

In an embodiment, control module 142 performs the steps of routine 206P.

In step 206Q, client 108 accesses an object pointer 206L in a document (object) table 206A. The object pointer is an element of the document table, which in one embodiment, is placed at the beginning of the content stream.

In step 206R, client 108 accesses a vtable pointer 206B for access to the vtable's function pointers 206G.

In step 206S, client 108 accesses an attribute pointer 206C for access to data in a content stream 206I.

In step 206T, client 108 uses the vtable pointer 206B to read function pointers 206G for access to instance methods 206H.

In step 206U, client 108 reads the content stream 206I for access to data 206K pointed to by attribute pointer 206C.

In step 206V, client 108 determines the requirements for the data 206K. In one embodiment, client 108 determines the amount of writeable memory that the data 206K will take up.

In step 206W, client 108 allocates writeable memory according to the requirements determined in step 206V.

In step 206X, client 108 decompresses and copies the data 206K into writeable memory. In one embodiment, the client 108 access portions of hardware and operating system software of device 106 to decompress and copy the decompressed data. In another embodiment, the client 108 internalizes the decompression and copying step.

In step 206Y, client 108 updates attribute pointer 206C to point to the data 206K in writeable memory.

In step 206Z, client 108 updates vtable pointer 206B to point to instance methods 206H for non-compressed, writeable data.

For example, referring also to FIG. 2F2, a routine can allocate writeable memory for data 206K2. It decompresses the data 206K and copies the data 206K into writeable memory, shown as data 206K2. The method then changes the attribute pointer 206C to point to data 206K2. It also changes the vtable pointer 206B to point to the non-compressed, writeable instance functions vtable 206D2.

With the modification complete, the instance method 206H calls an equivalent instance method 206H2 to actually perform the required write operation of the data 206K2.

A feature of this embodiment is that the operations of FIGS. 2F1 and 2F2 occur invisibly (transparently) to the application calling the pointers. An additional feature is that the operation is performed once on a per object basis.

Once performed, the object stays in writeable memory for the life of the document(s)/object(s).

A result of these embodiments is that object pointer 206L is preserved. This is important as many other objects may already have a reference to object 206A via object pointer 206L.

2.3 Detailed Object Model Embodiments

In an example application of the embodiment of FIG. 2F1, every object in the tree has an 8 byte (two pointers) entry in the Document Object Table (DOT). The entry consists of a pointer to the object's class vtable (table of function pointers for instance methods), and a pointer to the object's data. In the case of content stream formatted documents, the object's data pointer points into the content stream. It is noted that the invention is not limited to this example embodiment.

Inter-object references are via Object Identifiers (OID), which are unique 16 bit identifiers for each object in the document. Unlike pointers, OIDs support relocation and transmission between server and client. Entries in the DOT are ordered in OID order. That is, the OID is also the index into the DOT. Thus, an OID can quickly be translated into an Object pointer (which is desirable at runtime) by performing array arithmetic (object = &dot[oid]). By using OIDs and pointers in the DOT approach, a number of advantages are achieved: relocatability and transmittability from the OIDs, and ease of use, and a rational runtime model from the Object model.

The DOT is created when the document is being loaded, as shown in FIG. 2F1 and 2F2. An example scenario is as follows: The content stream is interrogated to see how many objects are in the document. A DOT large enough to accommodate that number of objects is created. An application quickly scans through the content stream, which has objects in OID order. For each object the application checks the type, ensures that the class and vtable 206D for that type

is created, writes the vtable pointer 206B in the DOT entry, then writes in the data pointer 206C, pointing back to the content stream data 206K. Then, the application skips to the next object in content stream (each entry has a length prefix stored in header 206J). When the application is done it has a fully populated DOT, with a number of objects that are writeable. If more objects are added to the document (via scripting requests), they are dynamically added to the end of the DOT in the form of FIG. 2F2, for example.

Like the PODS object model, the invention's Object Model, while implemented in C, is compatible with C++. Objects can be represented as abstract classes (pure virtual member functions), where the base class has no data slots. On most C++ compilers, this will guarantee that the C++ vtable entry will be forced to offset zero in the class. The benefit: both C [i.e., ADOM setValue(object, "xxx")] and C++ [i.e., object->setValue("xxx");] bindings (APIs) to the same object are provided.

Referring to FIG. 2H, a diagram illustrating content structure according to an embodiment of the invention is shown.

Document stream 208A illustrates at a high level the structure of the document that includes header 208B, string table 208C, and object table 208D. Header 208B includes the document table information, which can include object size, number of objects and string sizes. String table 208C includes string identifiers (SIDs) 208E that are ordered based on the SID in a manner similar to OIDs, as discussed above. String identifiers 208E include lists of word identifiers 208F, which are ordered word tables 208G. Each word table contains characters 208H that are represented by a certain code 208I.

The structure embodied in stream 208A provides the compressed and ordered nature described herein. Object table 208D includes the type and size and other attribute information for each object in the document as in FIG. 2E. String objects in these objects are string identifiers referring to 208C.

2.4 CSS Style Sheet Technology on Mobile Devices

Style Sheets represent a mechanism for setting and holding style attributes. HTML elements have a number of attributes that are stylistic in nature (dimensions, colors, font information, list bullet styles, etc.). Style Sheets are just a formalized scheme of setting, getting, and most importantly sharing these attributes. An advantage of Style Sheets is that they make it very easy to make global or semi global stylistic changes to a document.

For example, if one wants all the image borders to be 4 pixels wide, one can do that easily in one place. An advantage of supporting styles is it will make it easier for the content developer to share HTML between desktop and mobile devices - without recoding the HTML.

Also, style sheets represent a useful abstraction for attribute information. In an embodiment, the HTML Element super class of HTML/DOM is aware of style sheets. HTML Element provides the interface to get all attribute information for each object in the document. In one embodiment, HTML Element obtains most of this information from the style sheet that is associated with the object. As style sheets tend to be shared by like objects, the memory hit is not substantial.

FIGS. 2I1 and 2I2 demonstrate the effect of example style sheets.

In FIGS. 2I1 and 2I2, note the borders around Product List. The top and left borders are lighter, while the bottom and right borders are darker than the background - creating a stand-out 3D effect.

The tree is made up of a three element list, where each item is a title label and an embedded list. The embedded list is hidden by setting the top level display property to none, then shown again by setting the property to block.

FIG. 2J demonstrates floating images, where the text flows around the image. The example of FIG. 2J may represent a page created as a HTML source file, compiled on the server, and loaded onto the device for viewing.

2.5 Overview to the Architecture of the Technology

According to an embodiment, the architecture of the invention is designed so that a majority of the code is in the cross platform core pieces. A component that cannot be easily made cross platform is graphics code that draws bits on the screen. According to an embodiment, an approach is to break this component out as the Render abstraction, define a strict interface between Render and the rest of the code, and then utilize platform specific graphics subsystems capable of plugging back into the rest of the system.

The Draw code module deals with the semantic and programmatic level of graphics, leaving the Render module with only the responsibility of putting bits on the screen, such as bits in the form of strings, rectangles, border frames, etc. By keeping the abstraction level low, more functionality is maintained in the cross platform code, leaving the platform specific Render engine author with less code to write and maintain.

2.6 Classes and Relationships Within the Technology

The relationship between the render modules and the rest of the system is defined by a render interface. In one embodiment, an example header file can be implemented. Such an example header file can includes one or more of three interfaces: ARenderMgr, ADrawCtx, and ARenderFont.

In one embodiment, ARenderMgr is the render engine. In a given executable based on this technology, one can have 0, 1, or N objects that implement an ARenderMgr interface. In order to do a layout or draw on a DOM tree, one must have an ARenderMgr instance (actually one needs a DrawCtx which is created by a RenderMgr object - - see the discussion below). The render engine for a given platform's graphics subsystem is created by a factory method on the object subclass implemented by that platform.

In one embodiment, ARenderFont is a font in the world of this technology. ARenderFont basically defines a logical abstraction of the kinds of questions the core code (especially layout) needs to ask of a font, without locking down to the (very) platform specific details of any platform's font subsystem. ARenderFont only defines font metrics, such as "in one font, how many pixels wide is this string," but most platform specific render engines may sub class and add drawing behavior to the ARenderFont class.

In one embodiment, ADrawCtx is a drawing context and includes many rendering and drawing capabilities. ADrawCtx provides methods for getting bits on the screen (drawString(), drawRectangle(), etc), it provides the abstraction for the logical drawing surface (the Window, Form, or Port) that drawing occurs on, and also acts as memo pad for coordinate information during drawing.

2.7 Example Features

Cross-platform construction, layout, and rendering technology for DOM application browsers or viewers for mobile devices and DHTML layout and cascading style sheets

According to an embodiment of the invention, the cross-platform construction, layout, and rendering technology first parses HTML and then constructs a document object model based on HTML tags. It then lays out those objects in the tree that represent those tags in their logical pattern relative to their parent and children including details such as an x/y location and width and height attributes. After the layout is complete, the invention serializes the data and transmits it to the client in the case of the client/server browser embodiment.

FIG. 2D1 and 2D2 (described above) illustrate example diagrams showing an embodiment of the rendering of the client and/or server.

Rendering functionality can reside on device or on the server. All of the processes can be present on the server, leaving only the device specific reader to complete the pixelation of the objects, or all of the processes can be present on

the device including the parser. Thus, in some embodiments, the device can receive HTML and parse it appropriately.

FIG. 2K illustrates an example construction, layout, rendering and cross-platform technology architecture according to embodiments of the invention (see also FIGS. 2D1 and 2D2).

The diagram 227 includes server 104 and device 106. As discussed above, content 227A in the form of HTML, CSS, wireless markup language (WML), or other format enters the system.

Parser module 227B receives the content and formats it into discrete objects. Parser 227B assembles the objects into DOM 227F. Here, DOM 227F is simply a placeholder for the content stream as it is being assembled. Layout module 227C reads configuration information from proxy render module 227E. Proxy render module 227E obtains this information from the device, shown here as device configuration 227D.

Once the content stream has been assembled, emitter and compressor module 227G forwards the content stream via communications medium 227H to client 108.

The client 108 receives the content stream in DOM 227L. Loader 227J forwards the content stream to render module 227M for display on screen 227N.

Additionally, client 108 can also include optional parser module 227I, and optional layout module 227K. In embodiments that include these modules, client 108 is capable of receiving content 227A and parsing it into a content stream. Furthermore, layout module 227K can provide the proper configuration information directly from the device on which it is operating.

In an embodiment, the above-described processes begin with a user account and device specific sync to server 104, where the server 104 preemptively gathers enough information to represent the requirements of device 106.

If the device is capable of layout (i.e., includes a layout and rendering module 134), then the server may sync to the device only the DOM in content

stream format. Layout operations are then performed on the device for the document.

If the device does not have a layout module, or if the server is configured to perform the layout operations, then the server will layout the document and then sync the document content and layout information to the device.

Another feature of this embodiment is that HTML is sent to the server. There a parser communicates with the document object model that creates a collection of objects (including the ability to generate objects based on invalid HTML). From these objects, a tree of objects is generated that includes their layout attributes such as x/y coordinates and their width and height. This layout is based on a proxy of the device on the server.

Another feature is that the level of the technology that is synced or loaded to the device can be preemptively determined. The device, known or unknown to the server, will sync a proxy, or map, of the device requirements to the server. At the server, objects are rendered to the specifications of that proxied device before the device syncs. This involves parsing, rendering, and laying out of document objects using the proxy map of the device.

For example, suppose that a future device syncs to a current server. The invention enables the device to inform the server of enough information to represent its requirements. In other words, the device provides enough information to enable the server to create a proxy of the device. The server then proxy-renders the objects to the device based on the proxy of the device. Then, the server may send the parsed and laid out objects or the parsed only objects to the device. In the case of raw HTML being sent to the device, the server may not need to participate in the parsing process or other processes performed by the client.

FIG. 2L illustrates an example process where a future device is able to sync with a current server. The loader, layout, and rendering modules (as described above with respect to FIG. 2K) of the client employ an incremental and/or on-demand approach. To view the first page of the document, only the

first few objects (those visible) are loaded from the content stream. Content stream layout and rendering operations stop after the visible set of objects has been handled.

On-device pages are not necessarily created in their entirety (although they can be). They may be viewed as instantiated objects as they are needed. The content stream is rendered based on the need for the elements on a page. Compressed (in content stream format) objects are instantiated based on deltas against a default set of attributes (such as found in HTML) coded into the client application.

For example, if a document sent to the device needs to be viewed, it is quickly rendered as a set of objects rather than as a rebuilt page of HTML.

In addition, if the page needs to change, the relevant objects are incrementally decompressed from the content stream as described above with respect to FIGS. 2F1 and 2F2. Thus, the decompressed versions of only the required objects are created. When a value in an attribute is changed by a user (for example, the name of a button is lengthened or a paragraph of text is added to the page), only the objects that must be changed are decompressed.

For each instance of these objects, there is a table of objects providing an efficient way of making a byte stream into a data model. The tables are heuristic tables for matching calls to data. The objects viewed are seen through the laid out objects. When the objects are changed, a duplicate is created or morphed from the original object or set of objects and thus expanded based on a set of deltas. The result is a fully rendered copy of the object that is now writeable. A key to this functionality is how small the viewable objects are (as they need to be on mobile devices) until they need to be writeable. They become writeable when they are uncompressed or inflated to a fully rendered version.

The document object table that enables the content stream to be rendered by the invention is variable and customizable. The table may map to a function that calls data as easily as it maps to tables of data as easily as it maps to strings of code.

Another feature is that of on-device HTML authoring. Rendered HTML and resources can be redrawn and the HTML can be sent back in a content stream via a sync for storage in a database or posting on a network, intranet, or Internet.

Another feature is that the proxy-rendered nature of the invention enables a sync process to "take-over" a device. The result may be a single function device with a browser or application lock that can only be reset by another sync to the server, based on new server preferences. This means the client in conjunction with the object renderer and via a sync or other "install" mechanism, can "take over" a device for a use as a single function/single application device.

3.0 Server-Side Preparation of Data, HTML, and Other Network Resources and Objects for Ease-of-Use on Mobile Devices

3.1 Server Cache Operations

As previously stated, in embodiments the invention uses server logic to optimize content for delivery on mobile devices. Server 104 also stores optimized content in a cache. The optimized content is based on the type of mobile device that requested the content. Thus, the invention stores device specific versions of content requested by mobile devices. For example, suppose a first Palm device requests document A from provider 128A. Provider 128A controls page caching using relative or absolute date-time stamps. Server 104 may optionally override the page caching from Provider 128A. Document A is retrieved from provider 128A and optimized by web synchronization module 124 for use on the first Palm device. The optimized content is then cached for the first Palm device. Next a Windows CE device requests document A from provider 128A. Document A is retrieved from provider 128A, optimized for use by the Windows CE device, and cached for the Windows CE device. If, sometime later, a second Palm device with identical or similar characteristics (depending on the implementation) as the first Palm device requests document A from provider 128A, document A, specific to the first Palm device, is

immediately retrieved from the cache. The web synchronization module 124 does not have to retrieve document A from provider 128A for the second Palm device. As the number of users increases, the cache hit ratio increases, resulting in fewer fetches from providers 128. As the need for web synchronization module 124 to retrieve an object from provider 128 decreases, server bandwidth requirements also decrease.

FIG. 3A is a flow diagram representing an exemplary server cache operation of transformed content. The process begins with step 302. In step 302, a data object request is made by a mobile device. The mobile device may be, but is not limited to, any mobile device listed in Table 2. The process proceeds to step 304.

In step 304, web synchronization module 124 checks to see if the data object specific or applicable to the requesting mobile device is found in the cache associated with the server. If the data object specific or applicable to the requesting mobile device is found in the cache and is still valid, the process proceeds to step 306.

In step 306, web synchronization module 124 retrieves the optimized data object from cache. The process then proceeds to step 314.

Returning to step 304, if the data object specific or applicable to the requesting mobile device is not found in the cache or is no longer valid, the process proceeds to step 308.

In step 308, web synchronization module 124 retrieves the data object from provider 128. The retrieved data object may include a date-time stamp and/or other information indicating when the data object will expire. The process then proceeds to step 310.

In step 310, web synchronization module 124 transforms the data object to a form suitable for use and/or display on the requesting mobile device. For example, an address book entry will differ for Palm and Windows CE devices. The process then proceeds to step 312.

In step 312, the transformed data object is stored in the cache with device specific information, along with information on how long the data object can remain in the cache (such information may include the date-time stamp information of step 308). Note that the transformed data object is only stored in the cache if information retrieved from provider 128A indicates that the data object is cacheable or if server 104 is set to override the information from provider 128A. The process then proceeds to step 314.

In step 314, the transformed data object is sent to the requesting mobile device.

3.1.1 Special Case of Optimization for Mobile Devices: Negative Caching of Compressed Error Messages

In one embodiment, negative caching is implemented. Negative caching involves caching errors that result when web synchronization module 124 is unable to retrieve the requested data object from provider 128. This eliminates the need to subsequently access the provider 128. For example, if provider 128 is down, negative caching may lessen the number of negative hits that would otherwise result if attempts were made to retrieve data objects from provider 128. When web synchronization module 124 subsequently checks the cache for the irretrievable requested data, it will see the cached error message and will make no attempt to retrieve the data from provider 128.

For example, suppose device A requests a page. The web synchronization module 124 requests the page from provider 128. However, when web synchronization module 124 requests the page from provider 128, an error is returned indicating that the page is unavailable. Web synchronization module 124 caches the error, setting configurable expiration information. Later, another device (device B) with characteristics similar to device A requests the same page. Web synchronization module 124 will see the cached error (assuming that this cached error is still valid) indicating that the page is irretrievable. Therefore, web

synchronization module 124 will not have to make an attempt to retrieve the page from provider 128.

FIG. 3B is an exemplary flow diagram representing negative caching of error messages. The process begins with step 322. In step 322, a data object request is made by a mobile device. The process proceeds to step 324.

In step 324, web synchronization module 124 checks to see if the data object specific or applicable to the requesting mobile device is found in the cache associated with the server. If the data object specific or applicable to the requesting mobile device is found in the cache, the process proceeds to step 326.

In step 326, web synchronization module 124 retrieves the optimized data object from the cache. The process then proceeds to step 336.

Returning to step 324, if the data object specific or applicable to the requesting mobile device is not found in the cache, the process proceeds to step 328.

In step 328, web synchronization module 124 attempts to retrieve the data object from provider 128. The retrieved data object may include a date-time stamp and/or other information indicating when the data object will expire. The process then proceeds to decision step 330.

In decision step 330, it is determined whether an error indicating that the requested data object was irretrievable occurred as a result of step 328. If an error message occurred, the process proceeds to step 333. If no error message occurred, the process proceeds to step 332.

In step 332, web synchronization module 124 transforms the retrieved data object to a form suitable for use and/or display on the requesting mobile device. The process then proceeds to step 334.

Returning to decision step 330, if an error message occurred, web synchronization module 124, in step 333, transforms the error message to a form suitable for display on the requesting mobile device. The process then proceeds to step 334.

In step 334, the transformed data object or the error message is stored in the cache with device specific information, which may include an indication of how long such information may be cached (as explained above). Note that the transformed data object or error message is only stored in the cache if information retrieved from provider 128A indicates that the data object is cacheable or if server 104 is set to override the information from provider 128A. Server 104 may provide special overrides for negative caching of error messages. The process then proceeds to step 336.

In step 336, the transformed data object or the error message is sent to the mobile device.

3.1.2 Stochastic Cache Expiration Algorithm

Normally, when a large group of users sync daily, at least some of the users are syncing the same version of a set of pages all of which will expire at the same time. For example, if server 104 has a million users with a page on each user's device that expires at 12 midnight on September 9, 2000, every single device connected to server 104 at that moment (which in a wireless world may be all of the mobile devices) will request server 104 to provide those pages. An example diagram illustrating all hits on a server occurring at the same time is shown in FIG. 3C for the above example. As seen in FIG. 3C, the diagram shows all of the hits on server 104 occurring at once, that is, 12 midnight. This causes slow serving of new pages to all devices and puts excess stress on server 104 and provider 128.

To prevent the above scenario from occurring, an embodiment of the invention randomizes the expiration of the objects. This results in fast serving of new objects to all devices and puts less stress on server 104 and provider 128.

Server 104 sets a freshness lifetime for each object (or, in some embodiments, for groups of objects) stored in the cache. If the age of an object stored in the cache is within some % of the freshness lifetime (e.g. if it is about

to expire), otherwise known as Server FL or server freshness lifetime, then server 104 will vary the expiration of the cached object to determine whether the cached object should expire. The % of the freshness lifetime is usually set at the startup of server 104 using server preferences set by an administrator. Alternatively, the % of the freshness lifetime may be configurable.

Server 104 uses freshness lifetime for determining whether or not to modify the expiration of the cached object. Server 104 determines whether the cached object is close to expiring and then in an embodiment it uses a stochastic function to determine whether or not to expire the cached objects. The stochastic function is used if the current age of the cached object is within some percentage of the freshness lifetime.

FIG. 3D is an example flow diagram representing a method for randomizing the expiration of objects set to expire at the same time according to an embodiment of the invention. The process is performed for any given object in the cache (called the cached object). The process begins with step 342. In step 342, server 104 determines the cached object's freshness lifetime. The cached object's freshness lifetime (FL) is the time of expiration (Texp) of the cached object minus the time in which the object was placed in the cache (Tin-cache). In embodiments, the time of expiration is given when the object is retrieved from the provider 128. If the time of expiration is not given, server 104 may set a time of expiration for the object, or this may be configurable. The cached object's freshness lifetime is:

$$\text{Object's FL} = \text{Texp} - \text{Tin-cache} \qquad (1)$$

The process then proceeds to step 344.

In step 344, server 104 determines the cached object's age. The cached object's age is the total time that the object has been stored in the cache. The cached object's age is

$$\text{Object's Age} = \text{Tnow} - \text{Tin-cache} \qquad (2)$$

where Tnow is the current time. The process then proceeds to step 346.

In step 346, server 104 determines the % of the object's freshness lifetime. The % of the object's freshness lifetime is the object's age divided by the object's freshness lifetime. The % of the object's freshness lifetime is:

$$\%\text{ of the Object's FL} = \text{Object's Age}/\text{Object's FL} \qquad (3)$$

The process then proceeds to decision step 348.

In decision step 348, it is determined whether the % of the Object's FL < % of the server FL. As previously stated, the % of the server FL is the % of the freshness lifetime set at the startup of server 104 using server preferences set by an administrator. Alternatively, the % of the server FL may be configurable. If it is determined that the % of the Object's FL is less than the % of the Server FL, then the process proceeds to step 350.

In step 350, the object is determined not to have expired. The process then proceeds to step 352.

In step 352, the object is retrieved from the cache and processed in an implementation or application dependent manner.

Returning to decision step 348, if it is determined that the % of the Object's FL is greater than or equal to the % of the Server FL, then the process proceeds to step 354.

In step 354, in an embodiment, a random number is generated using a random number generator. In one embodiment, the random number generator may be normally distributed. In another embodiment, the random number generator may be uniformly distributed. One skilled in the relevant arts would know that other distributions may be used without departing from the scope of the present invention. The process then proceeds to decision step 356.

In decision step 356, it is determined whether the random number generated in step 354 is less than the % of the Object's FL. If the random number is greater than or equal to the % of the Object's FL, then the process proceeds to step 350, where the object is determined not to have expired, and is then retrieved from the cache in step 352.

Returning to decision step 356, if it is determined that the random number generated in step 354 is less than the % of the Object's FL, then the process proceeds to step 358.

In step 358, it is determined that the object has expired. The process then proceeds to step 360.

In step 360, server 104 attempts to retrieve the object from provider 128.

The following is an example of how the above method works according to an embodiment of the invention. As previously stated, if the % of the Object's FL is within the % of the FL set by the server, the stochastic process is implemented. If, for example, an object's age is 190 and the object's freshness lifetime is 200, the object's age is 95% of the object's freshness lifetime. Suppose server 104 was set for a threshold of 90% of the freshness lifetime. When a decision is made to vary the expiration for determining whether the object is fresh or expired, the probability that the object is expired is determined by figuring out what percentage of the vary range (90-100%) the current age of the object has covered. The vary range is 90 - 100 percent, and since the age of the object is 95%, 50% is the probability that the object is expired. If the age were 92% of the freshness lifetime, the probability would be 20%, if 98, 80%, etc. Similarly, if the range were 80 - 100, and the age is 95%, the probability would be 75%. Then the random number is generated, and compared to the probability in order to determine if the object is fresh or expired. Table 3 shows the age of the object, the object's % of freshness lifetime, and the probability that the object will expire for the above example.

TABLE 3.

Age	% of Freshness Lifetime/100	Probability of Expiring
179	.895	0% (not considered)
185	.925	25%
190	.95	50%
195	.975	75%
200	1.0	100%

FIG. 3E is a diagram showing freshness lifetime for the object described in the above example. The shaded area indicates the vary range in which it is determined if the object is fresh or expired.

3.2 Syncing Mobile Devices

3.2.1 Single Account/Profile-Multiple Devices

Server 104 enables a single account/profile user to sync multiple devices and obtain device specific content on each device. FIG. 3F is an example block diagram 362 illustrating a single account/profile having multiple devices. Block diagram 362 comprises a user account/profile 364, a first mobile device 366, a second mobile device 368, a third mobile device 370, server 104, and providers 128. First mobile device 366 may be a Palm device. Second mobile device 368 may be a cell phone. Third mobile device 370 may be Windows CE device. Although FIG. 3F shows three mobile devices associated with a single account/profile, one skilled in the relevant art(s) would know that more devices or one less device could be added or subtracted, respectively, without departing from the scope of the invention. User account/profile 364 is associated with each mobile device 366-370. Each mobile device 366-370 interacts with server 104 via a transmission medium which may be any wired or wireless medium using any communication protocol. Server 104 may also connect to providers 128.

When any one of mobile devices 366-370 initially connects to server 104, each device 366-370 will provide its device characteristics to server 104. Server 104 will store the device characteristics in database module 126 and send the device characteristics to web synchronization module 124 and/or server extension module 156. For subsequent connections with server 104, each mobile device 366-370 will identify itself to server 104 and server 104 will retrieve the device's characteristics from database module 126. In one embodiment, as long as the user continues to sync devices 366-370 with server 104, server 104 will maintain each device's information on database module 126. If any one of devices 366-370 are not synced within some predetermined period, server 104 may optionally delete that device's information from database module 126. If a user syncs devices 366-370, one right after the other, each device 366-370 will have the same content, but the content will be optimized for each specific device 366-370.

An embodiment of the invention also provides a common link to share and sync data objects between disparate user devices. For example, if user account/profile 364 has a personal digital assistant and a cell phone, neither the PDA nor the cell phone may have the ability to communicate directly with each other. Assume an address book exists on both the cell phone and the PDA. The address book must be separately updated since the two devices do not have the ability to communicate with each other. However, by using server 104 of the present invention, the PDA and the cell phone may sync up with server 104 to provide the same information on both devices. In this example, the address book of both devices may sync up with server 104 to enable the address book on both the PDA and cell phone to contain the same information. Thus, the invention syncs disparate user devices by providing server 104 as a common link to share and sync data objects.

3.2.2 Single Device – Multiple Servers

The invention also enables a single device to connect to multiple servers. FIG. 3G shows an example screen shot for enabling a user to add multiple servers. FIG. 3H is an exemplary block diagram 363 representing a single mobile device that connects to multiple servers. Diagram 363 comprises single user account/profile 364 connected to mobile device 366, which, in turn, can be connected to a plurality of servers 365, 367, and 369 that may be connected to any variety of providers 128, such as the Web, a database, such as LOTUS Notes, or a network, respectively. Although FIG. 3H shows three servers 365-369, one skilled in the relevant art(s), based on the teachings contained herein, would know that more servers or less servers could be implemented without departing from the scope of the invention. When a user initiates a sync, in an embodiment, device 366 will be synced to each enabled server on device 366, one at a time.

FIG. 3I is an exemplary flow diagram representing a sync process for a device connected to multiple servers. The process begins with step 374. In step 374, a user initiates a sync. The process proceeds to step 376.

In step 376, a list of sync servers and accounts per server are retrieved. The process then proceeds to step 378.

In step 378, device 366 is synced to each server, one at a time.

3.2.3 Multiple Devices – Multiple Servers

The invention also allows multiple devices for a single user account/profile to be connected to a plurality of servers. A multiple device – multiple server scenario is shown in FIG. 3J. Although three mobile devices and three servers are shown in FIG. 3J, one skilled in the art would know that more or less devices and servers could be used without departing from the scope of the invention. In one embodiment, user account/profile 364 can cache a variety of device characteristics 366, 368, and 370 used to access a variety of servers 365,

367, and 369. In one embodiment, the user can store the device profiles on a desktop and store the multi-server connections on devices 366, 368, and 370. When the user initiates a sync for any one of devices 366, 368, and 370, a list of sync servers and accounts per server is retrieved for the device and the device is synced to each server in the list, one at a time. The user may then sync the next device, as described above in section 3.2.1.

4. Executing Scripts on Mobile Devices

4.1 Overview

According to an embodiment, the present invention transforms, loads, and executes scripts onto mobile devices. The present invention is able to retain the interactivity of the scripts with a data object, such as a HTML document or object of other forms, from which they were originally located. The present invention retains the functionality of the scripts on mobile devices, such as device 106. According to embodiments of the present invention, scripts can be executed on clients 108 which are currently connected to a server or network. According to alternative embodiments of the present invention, scripts can execute on devices which are not currently connected to a server or network through any type of connection (wired, wireless, etc.).

The terms "script," "executable script," "applet," and the plural form of these terms are used interchangeably throughout this document to refer to programs that accompany or are embedded within an object (It is noted that the invention is described, at times, in terms of HTML documents for illustrative purposes, but the invention is not limited to this examples. The invention includes operations with other types of data objects, such as but not limited to other mark-up language or tag-based documents.). Various HTML specifications define how scripts can be included in objects (i.e., Web pages), so that programmers can develop objects that can accept scripts.

Scripts can be programmed to execute immediately when the object is loaded into a client, a client's Web browser or other display-interface program. Alternatively, scripts can also be programmed to execute in response to browser events, such as when a user clicks on a button or when the user resets a form in the object.

The terms "scripting language," "script language," "programming languages," and the plural form of these terms are used interchangeably throughout this document to refer to computer programming languages used to make scripts. As one skilled in the relevant art(s) would recognize, based at least on the teachings herein, many programming languages can be used as scripting languages, including but not limited to the following programming languages: JavaScript, ECMAScript, Java, Perl, Tcl, Visual Basic, and VBScript.

4.2 Server-Side Compilation

Many programming languages are implemented with executable programs which transform their source file (e.g., a script) into a program. Many programming languages include a compiler. Some programming languages include an interpreter.

A compiler translates a file written in a source language into an equivalent computer program in a target language. Some compilers translate the file written in the source language directly to a machine language, which can be directly executed by a device 106, server 104, or client 108. Other compilers translate the file into an intermediate stage known as a virtual machine language.

An interpreter is an auxiliary program which can execute virtual machine code. Virtual machine languages are sometimes called byte code or P-code languages. Interpreters for these languages are sometimes called byte code interpreters or P-code interpreters.

FIG. 4 is a block diagram illustrating script compiler modules according to an embodiment of the invention. In FIG. 4, block diagram 400 depicts content

provider 128, server 104 and client 108. The server 104 includes script compiler modules 402a – 402n for compiling, transforming, and recompiling scripts into languages executable and compatible with client 108 and device 106. The client 108 includes several language interpreter modules 404a – 404n for interpreting scripts arriving in form of virtual machine language.

In FIG. 4, a content provider 128 sends scripts along with an HTML page to the server 104. The server 104 interprets the page's scripts, which generate HTML and scripts as its output. This output is tokenized and sent to the client 108.

The client 108 also includes several script compiler modules 406a – 406n which are capable of performing the functionality of script compiler modules 402a – 402n. Modules 406 are available in embodiments where the server 104 is bypassed.

The content provider 128 sends objects and scripts to the server 104. According to one embodiment of the present invention, the script compiler module 402 compiles the scripts to virtual machine language according to the specifications of the client 108 and/or device 106. The server 104 sends the compiled scripts along with tokenized object to the client 108.

In one embodiment of the present invention, the client 108 requests an object from the server 104. Upon receiving the request, the script compiler modules 402 of the server 104 compile each script associated with the object, according to the script language for which they are implemented. In embodiments of the present invention, the server 104 can contain a separate script compiler module for each scripting language which it supports.

In one embodiment, as shown in FIG. 4, the script compiler modules 402 are managed by the server extension module 156. Each compiled script is sent to the requesting client where it can be stored and executed. In alternative embodiments of the invention, the client-side script compiler modules 406 can compile scripts directly to the machine language of a target client 108. In other embodiments of the invention, the client-side compiler modules 406 can compile

scripts to a virtual machine language, which is interpreted by the language interpreter modules 404 of the client 108.

In one embodiment of the present invention, clients 108 and devices 106 can support several different virtual machine languages. A single client 108 can contain several language interpreter modules 404. The language interpreter modules 404 of client 108 can be managed by the client extension interface 138, as shown in FIG. 4.

In further embodiments, the server 104 can communicate with a number of clients 108 and devices 106 which support differing sets of virtual machine languages.

In one embodiment of the present invention, when the client 108 syncs with the server 104, it sends to the server 104 an encoded list of the virtual machine languages which it supports. In one embodiment of the present invention, if the server 104 is not able to compile a script in a particular script language to any of the languages which the client 108 supports, then it will not send any virtual machine code for the script to the client 108 and the client 108 can effectively ignore the script.

4.3 Storing Interpreter Data Structures

As described herein, the language interpreter modules 404 on the client 108 interpret virtual machine code sent from the server 104. In one embodiment, the client operating system (e.g., Palm OS, Windows CE, Pocket PC) provides applications with two different types of memory architecture: one type which is fast to write to; and another type which is much slower to write to (It is noted that these speeds are relative and the actual speeds are implementation and application dependant.).

For example, but without limitation, on Palm devices, Palm OS provides applications with dynamic memory, which is a read-write random-access memory (RAM) which can be read or written in a single machine instruction. It also

provides a storage memory, which is RAM which can be read with a single machine instruction, but which can be written only through a call to the operating system, which can take tens or hundreds of machine instructions to execute.

Here, the generic terms fast-write memory and slow-write memory are used. On Palm OS, dynamic memory is fast-write memory and storage memory is slow-write memory. In embodiments, language interpreter modules 404 store data structures which are frequently updated in fast-write memory, and store other data structures in slow-write memory.

According to embodiments of the present invention, certain virtual machine languages can contain instructions which manipulate objects. Each object has a number of properties, each of which has a name and a value. For example, the JavaScript language includes objects which have a name and a value. In embodiments using these or similar properties, language interpreter modules 404 store the name of each object property in slow-write memory. Modules 404 also store the value of each object property in fast-write memory. This means that the language interpreter modules 404 can update the value of an existing object property by writing only to fast-write memory. In one embodiment, adding a new property to an object requires writing to slow-write memory.

FIG. 5 is a block diagram illustrating a language interpreter module according to an embodiment of the invention. In FIG. 5, a diagram depicts how a language interpreter module 404 might represent a particular object O 502. Object O 502 has five properties, whose names 504 are "x," "y," "abc," "foo," and "xyz," respectively. The properties' values 510 are "3," "7," "home," "start," and "false," respectively.

As one skilled in the relevant art(s) would recognize, based at least on the teachings described herein, object O 502 can have more or less than five properties (i.e., more than or less than five names and/or values). More specifically, one skilled in the relevant art(s), based at least on the teachings

described herein (e.g., with respect to the discussion of FIGS. 5 and 6), would be able to construct object with any number of properties.

In one embodiment of the present invention, the base data structure representing the object O 502 is stored in slow-write memory and has two fields names 504 and values 510. The names 504 are used as pointers to an array in slow-write memory which holds pointers to each property name. The values 510 are pointers to an array in fast-write memory which holds each property value.

4.4 Using Object Delegation to Share Global Variables Across Pages

In one embodiment of the present invention, virtual machine languages can support global variables, which are values that are referenced by name anywhere within any script.

Delegation can be implemented in a number of ways. According to a method used in embodiments of the present invention, encapsulated objects are linked to other objects, so that they inherit properties via a parent or root object. Delegation introduces dynamic sharing of behavior between objects by adding the idea of a prototype or super object to every object. This in turn means that one can have a prototype or root object to begin process and pass along properties to all other linked objects.

FIG. 6 is a block diagram illustrating the operations of delegated language interpreter modules according to an embodiment of the present invention. In FIG. 6, diagram 600 illustrates the operations of language interpreter modules with delegation using represented global variables as the client 108 moves from one object to the next. There are two permanent global variables Math 608 and Date 610. These are stored as properties of the prototype object P 606.

On the first page represented by object G 602, the global variable document 612 points to an object representing the HTML document 618 viewed on the client 108. A script constructs two global variables x 616 and y 614 with values 5 and 7, respectively. The global variables document 612, x 616 and y 614

are stored as properties of a global object G 602 whose prototype object is object P 606. These variables are accessible to scripts only while the client 108 remains on the first page represented by object G 602. When the client 108 moves to a new HTML document, it constructs a new global object H 604 and gives it its own property document 614' pointing to the new page's document object 620. A script on the second page constructs two global variables x 616' and y 614' with values 4 and abc, respectively. These global variables are stored as properties of object H 604.

According to embodiments of the present invention, the values of certain global variables can depend on the Web page currently being viewed. For example, in some embodiments, a global variable document 612 contains an object representing the current HTML document, such as document object 618, which is different for each Web page view on client 108. In an alternative embodiment, scripts can define new global variables which should be accessible to other scripts on the same HTML document, but which should not be accessible to scripts running on other HTML documents.

In further embodiments, global values are made permanent. They have values which are the same for all HTML documents. For example, in embodiments which support the JavaScript language, there is a global variable Math which holds an object containing various mathematical utility functions, such as Math.sin and Math.cos. In this embodiment, all scripts on all documents can access the Math object.

The embodiments described above illustrate the use of virtual machine languages with instructions which manipulate objects with properties. In a language with delegation, each object can contain a pointer to a prototype object. In one embodiment, whenever the language interpreter module 404 looks for a property of an object O 502 , it first looks for the property in the object O itself, then in O's prototype object, then in that object's prototype object, and so on until it either finds the property or reaches an object which has no prototype object.

In one example, the Self language is used because it supports delegation. In another example, JavaScript is used because it also supports delegation. As one skilled in the relevant art(s) would recognize, based at least on the teachings described herein, other languages can be used, so long as they support the features of delegation.

In embodiments of the present invention, where the script language is the JavaScript language, global variables are actually properties of a global object. In one embodiment, if a virtual machine language supports delegation, then the global object G 602 has a prototype object P 606. The object G 602 contains global variables whose values depend on the HTML document currently being viewed.

The object P 606 contains global variables whose values are the same for all HTML documents. When the browser moves to a HTML document, the virtual machine interpreter modules 404 construct a new global object H 604. Object H's prototype object is the same object P 606 which was used as the prototype object for the old global object G 602. This means that the interpreter module 404 does not need to reconstruct the global values in object P 606 or add them explicitly to the new global object H 604. The global values in object P 606 are accessible through object H 604 just as they were through object G 602.

While only object 602 – 606 are described, it will be apparent to one skilled in the relevant art(s), based at least on the teachings herein, that the system of the present invention can support a plurality of objects and HTML documents. Furthermore, according to the embodiments described herein, the objects can be cascaded with additional layers of global objects.

4.5 Server-Side Interpretation

Referring to FIG. 7, according to one embodiment of the present invention, the server 104 can contain one or more server-side language interpreter

modules 704 which are used for interpreting scripts. Modules 704 are managed by the server extension module 156.

In an embodiment, if a device 106 requests an object containing a script; and

- the device does not support any virtual machine language to which the server can compile the script;
- the server contains a server-side language interpreter module which can interpret the script;
- the script requires no user input;
- the script performs output only by generating HTML;
- the script requires no access to objects which are available only when its HTML page is viewed in a browser;

then the server 104 interprets the script, and sends the resulting HTML to the device 106 without sending the compiled script to the device 106. This allows even devices 106 and client 108 which have no virtual machine interpreter modules 404 to view and interact with objects which contain scripts.

FIG. 7 is a block diagram illustrating a server with a server-side language interpreter module according to an embodiment of the invention.

More specifically, FIG. 7 depicts a server 104 with a server-side language interpreter module 704 communicating with a client 108 that has no language interpreter modules 404. In FIG. 7, a content provider 128 sends scripts along with an HTML page to the server 104. The server-side language interpreter module 704 interprets the page's scripts, which generate HTML as their output. This HTML is tokenized and sent to the client 108 along with the HTML representing the page itself.

4.6 Further Embodiments of Script Operations

Referring to FIG. 8, a routine for script operations according to an embodiment of the invention is shown. The process begins with step 802 and

proceeds to step 804. In step 804, a client 108 sends a request for a document to a server 104. The client 108 can be on a device 106. The process proceeds to step 806.

In step 806, the client 108 sends a list of languages that it supports (i.e., languages that is understands and is able to compile and/or interpret) to the server 104. Although, at times, this embodiment refers to documents, the invention operates with any objects. The process proceeds to step 808.

In step 808, the client 108 receives from the server 104 the document that it requested along with a script related to that document. The client 108 can request more than one document, such as a series of documents which constitute a Web site. Accordingly, the client 108 can receive any number of scripts related to those requested documents. The process proceeds to step 810.

In step 810, the client 108 stores the document and script that it received in step 808. The process proceeds to step 812.

In step 812, the client 108 accesses the stored document. The process proceeds to step 814.

In step 814, the client 108 executes the stored script related to the stored document. The process proceeds to step 816.

In step 816, the client 108 updates the properties and/or performs other functions as called on by the executed script.

While the above embodiment describes the client 108 as operating on one document and one script, one skilled in the relevant art(s), based at least on the teachings described herein, would recognize that the above embodiment can operate with any number of documents and scripts.

Referring to FIG. 9, a flowchart showing a routine for executing scripts according to an embodiment of the invention is shown. The process provides a detailed embodiment of the operations of step 814. The process begins in step 904. In step 904, the client 108 determines the language in which the script is written. The process proceeds to step 906.

In step 906, the client 108 performs the optional step of compiling the script. In one embodiment, script compiler module 406a compiles the script. The process proceeds to step 908.

In step 908, the client 108 interprets or executes the script. In one embodiment, language interpreter module 404a interprets the script. The process proceeds to step 910.

In step 910, the client 108 receives a new page specific global object. In one embodiment, the client 108 receives the new page specific global object from operations performed by the script's operation in step 814. The process proceeds to step 912.

In step 912, the client 108 forwards a new page specific global object to storage memory for later retrieval. The process proceeds to step 816.

Referring to FIG. 10, a flowchart showing a routine for updating properties according to an embodiment of the invention is shown. The process provides a detailed embodiment of the operations of step 816. The process begins in step 1004. In step 1004, the client 108 accesses properties associated with a new page specific global object. In one embodiment, the new page specific global object is determined in step 814. The process proceeds to step 1006.

In step 1006, the client 108 updates the associated properties. The process proceeds to step 1008.

In step 1008, the client 108 stores the associate properties.

Referring to FIG. 11, a flowchart showing a routine for server-side script operations according to an embodiment of the invention is shown. The process begins with step 1102 and proceeds to step 1104. In step 1104, a client 108 sends a request for a document or other object to a server 104. The client 108 can be on a device 106. The process proceeds to step 1106.

In step 1106, the client 108 sends a list of languages that it supports (i.e., languages that is understands and is able to compile and/or interpret) to the server 104. In one embodiment, the client 108 can send a null list to the server 104 that

indicates that the client 108 does not support any languages. The process proceeds to step 1108.

In step 1108, the client 108 receives from the server 104 the document or other object that it requested with information representing the related scripts. The client 108 can request more than one document or object, such as a series of documents which constitute a Web site. Accordingly, the client 108 can receive any number of pieces of information related to those requested scripts. The process proceeds to step 1110.

In step 1110, the client 108 stores the document and information that it received in step 1108. The process proceeds to step 1112.

In step 1112, the client 108 accesses the stored document. The process proceeds to step 1114.

In step 1114, the client 108 accessed the stored information related to the represented script.

While the above embodiment describes the client 108 as operating on one document or object, one skilled in the relevant art(s), based at least on the teachings described herein, would recognize that the above embodiment can operate with any number of documents.

5. Conclusion

While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example only, and not limitation. It will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined in the appended claims. Thus, the breadth and scope of the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.